Homotopy Type Theory

Lecture 1: Dependent Type Theories

Nicola Gambino Dan Licata Peter LeFanu Lumsdaine

Mathematical Structures of Computation

Lyon, January 2014

Homotopy type theory

- Dependent type theories are a well-known class of formal systems.
- A new class of models, based on ideas of homotopy theory, has been discovered around 2007-08.
- This has suggested a new way of formalizing mathematics in type theories and new type-theoretic axioms.
- The new dependent type theories suggest many questions.

Overview of the lectures

1. The syntax of dependent type theories
2. Homotopy-theoretic models
3. The univalence axiom
4. The computational interpretation of the univalence axiom
5. Higher inductive types, examples of synthetic homotopy theory
6. Agda/Coq formalizations

Dependent type theories vs first-order theories (I)

Steps to set up a first-order theory:

(1) fix a language L
(2) define inductively the set of well-formed terms
(3) define inductively the set of well-formed formulas
(4) give the axioms for the theory
(5) define inductively the set of theorems of the theory.

Note. Each step depends only on the previous ones (e.g. the axioms of a theory do not modify the set of well-formed terms).

Dependent type theories vs first-order theories (II)

Problem. This approach does not work for dependent type theories:

- deduction rules of a dependent type theory specify how the well-formed term expressions are built, e.g.

$$\frac{a : A}{\mathsf{inl}(A, B, a) : A + B}$$

- the sets of term and type expressions cannot be defined independently, e.g.

$$\mathsf{Id}(A, a, b)\,, \quad \mathsf{nil}(A)$$

A solution. Define dependent type theories within meta-theories known as logical frameworks.

Setting up a dependent type theory

Other solution (Aczel, . . .)

(1) fix a signature (*a notion that will be defined in the next slides*),
(2) define inductively the set of raw expressions,
(3) give the deduction rules of the dependent type theory,
(4) define inductively the set of theorems of the theory,
(5) the theorems isolate the well-formed expressions.

We will illustrate this in general and in one example.

Signatures for dependent type theories (I)

A signature for an algebraic theory (e.g. theory of groups) consists in:

- a set of operations,
- an assignment of an arity to each operation.

Here, an arity is just a natural number (the number of arguments of the operation).

For dependent type theories, we need more a complex notion of arity, to account for **variable binding** operations, e.g. λ , Π, Σ.

Note. Ongoing research on theories with variable binding.

Signatures for dependent type theories (II)

Definition. An **arity** is a tuple of the form

$$\big((n_1, \varepsilon_1), \ldots, (n_k, \varepsilon_k), \varepsilon\big)\,,$$

where $k \in \mathbb{N}$, $n_1, \ldots, n_k \in \mathbb{N}$ and $\varepsilon_1, \ldots, \varepsilon_k, \varepsilon$ are either 0 or 1.

Notation. When $k = 0$, we will write (ε).

Idea.

- An operation of arity as above takes k arguments and binds n_i variables in the i-th argument.
- The $\varepsilon_1, \ldots, \varepsilon_k, \varepsilon$ keep track of the distinction between term expressions (0-expressions) and type expressions (1-expressions).

Definition. A **signature** Σ is a set of pairs (s, α) where α is an arity. If $(s, \alpha) \in \Sigma$, we call s a **symbol of arity** α.

Raw expressions

Fix

- an infinite set of variables $\{x_0, x_1, \ldots\}$,
- a signature Σ.

Define the sets of 0-expressions and 1-expressions by the rules:

(i) every variable is a 0-expression,

(ii) if s has arity $((n_1, \varepsilon_1), \ldots, (n_k, \varepsilon_k), \varepsilon)$ and M_i is an ε_i-expression and $\vec{x}_i$ is a vector of n_i distinct variables for $i = 1, \ldots, k$, then

$$s\big((\vec{x}_1)M_1, \ldots, (\vec{x}_k)M_k\big)$$

is an ε-expression.

Notation. When $k = 0$, we write s rather than $s(\,)$. Also, if some $n_i = 0$ we write M_i rather than $(\,)M_i$.

Example

The signature for the type theory $\mathbf{ML_1}$ includes the symbols

- Nat of arity (1)
- succ of arity $((0,0),0)$
- nil of arity $((0,1),0)$
- λ of arity $((0,1),(1,1),(1,0),0)$

Thus, we have that

- Nat is a 1-expression
- $\mathsf{succ}(x)$ is a 0-expression
- $\mathsf{nil}(\mathsf{Nat})$ is a 0-expression
- $\lambda(\mathsf{Nat},(x)\mathsf{Nat},(x)x)$ is a 0-expression, usually written $(\lambda x : \mathsf{Nat})x$.

Judgements

A **judgement** has one of the following four forms:

- $A : \mathsf{type}$
- $A = B : \mathsf{type}$
- $a : A$
- $a = b : A$

Here, A, B stand for 1-expressions and a, b for 0-expressions.

Contexts and hypothetical judgements

A **context** is a sequence of the form

$$x_0 : A_0 \, , x_1 : A_1 \, , \ldots , x_n : A_n$$

where

- $x_0, \ldots, x_n$ are distinct variables
- $A_1, \ldots, A_n$ are 1-expressions.

Notation. When $n = 0$, we write $()$.

A **hypothetical judgement** has the form

$$\Gamma \vdash J$$

where Γ is a context and J is a judgement.

Notation. We write J instead of $() \vdash J$.

Deduction rules

A **deduction rule** has the form

$$\frac{\Gamma_1 \vdash J_1 \qquad \cdots \qquad \Gamma_n \vdash J_n}{\Gamma \vdash J}$$

where $\Gamma_1 \vdash J_1 \,, \ldots, \Gamma_n \vdash J_n \,, \Gamma \vdash J$ are hypothetical judgements.

Notation. When $n = 0$, we simply write $\Gamma \vdash J$.

A dependent type theory is given by specifying

- a signature,
- a set of deduction rules.

Derivability is defined in the standard way.

Well-formed expressions

Definition. Let T be a dependent type theory over a signature Σ.

- If $\Gamma \vdash A : \mathsf{type}$ is derivable, then we say that A is a **well-formed type** in context Γ.
- If $\Gamma \vdash a : A$ is derivable, then we say that a is a **well-formed element** of type A in context Γ.
- If $\Gamma \vdash A = B : \mathsf{type}$, then we say that A and B are **definitionally equal well-formed types** in context Γ.
- If $\Gamma \vdash a = b : A$, then we say that a and b are **definitionally equal well-formed elements** of type A in context Γ.

The dependent type theory $\mathbf{ML_1}$

A dependent type theory with the following forms of type:

$$\mathsf{0}\,,\quad \mathsf{1}\,,\quad \mathsf{Bool}\,,\quad \mathsf{Nat}\,,\quad \mathsf{List}(A)\,,\quad A+B\,,$$
$$\mathsf{Id}(A,a,b)\,,\quad (\Pi x:A)B\,,\quad (\Sigma x:A)B\,,\quad \mathsf{U}$$

Note.

- **ML** stands for Martin-Löf
- The subscript 1 indicates the presence of rules for one type universe

The signature of $\mathbf{ML_1}$ (I)

Symbol	*Arity*
$\mathsf{0}, \mathsf{1}, \mathsf{Bool}, \mathsf{Nat}, \mathsf{U}$	(1)
List	$((0,1),1)$
$+$	$((0,1),(0,1),1)$
Id	$((0,1),(0,0),(0,0),1)$
Π	$((0,1),(1,1),1)$
Σ	$((0,1),(1,1),1)$
T	$((0,0),1)$

Notation. We will write

- $A + B$ for $+(A, B)$
- $(\Pi x : A)B$ for $\Pi(A, (x)B)$
- $(\Sigma x : A)B$ for $\Sigma(A, (x)B)$.

The signature of $\mathbf{ML_1}$ (II)

The other symbols of the signature:

$*$	refl
unitrec	J
true	λ
false	app
boolrec	pair
0	split
succ	$\mathsf{Bool}^{\mathsf{U}}$
natrec	$\mathsf{Nat}^{\mathsf{U}}$
nil	0^{U}
cons	$\mathsf{List}^{\mathsf{U}}$
listrec	$+^{\mathsf{U}}$
inl	Id^{U}
inr	Π^{U}
case	Σ^{U}

The deduction rules of $\mathbf{ML_1}$

(1) General rules

(2) Rules for the forms of type of $\mathbf{ML_1}$. For each one, we have

- formation rules
- introduction rules
- elimination rules
- computation rules

Note.

- When stating a rule, we omit contexts that are common to premisses and conclusion.
- Sometimes deduction rules are given as schemes.

General deduction rules

Standard rules regarding context formation, substitution, equality.

Examples.

$$\frac{\Gamma, \Delta \vdash J \qquad \Gamma \vdash A : \mathsf{type}}{\Gamma, x : A, \Delta \vdash J} \; x \notin \mathrm{FV}(\Gamma) \cup \mathrm{FV}(\Delta)$$

$$\frac{x : A, \Gamma \vdash J \qquad a : A}{\Gamma[a/x] \vdash J[a/x]}$$

$$\frac{a : A \quad A = B : \mathsf{type}}{a : B}$$

The type of natural numbers (I)

Formation rule.

$$\mathsf{Nat} : \mathsf{type}$$

Introduction rules.

$$0 : \mathsf{Nat} \qquad \frac{n : \mathsf{Nat}}{\mathsf{succ}(n) : \mathsf{Nat}}$$

The type of natural numbers (II)

Elimination rule.

$$\frac{c : \mathsf{Nat} \quad u : \mathsf{Nat} \vdash E : \mathsf{type} \quad d : E[\mathsf{0}/u] \quad x : \mathsf{Nat}, y : E[x/u] \vdash e : E[\mathsf{succ}(x)/u]}{\mathsf{natrec}(c, (u)E, d, (x, y)e) : E[c/u]}$$

Computation rules.

$$\frac{u : \mathsf{Nat} \vdash E : \mathsf{type} \quad d : E[\mathsf{0}/u] \quad x : \mathsf{Nat}, y : E[x/u] \vdash e : E[\mathsf{succ}(x)/u]}{\mathsf{natrec}(\mathsf{0}, (u)E, d, (x, y)e) = d : E[\mathsf{0}/u]}$$

$$\frac{c : \mathsf{Nat} \quad u : \mathsf{Nat} \vdash E : \mathsf{type} \quad d : E[\mathsf{0}/u] \quad x : \mathsf{Nat}, y : E[x/u] \vdash e : E[\mathsf{succ}(x)/u]}{\begin{array}{l}\mathsf{natrec}(\mathsf{succ}(c), (u)E, d, (x, y)e) \\ \quad = e[c/x, \mathsf{natrec}(c, (u)E, d, (x, y)e)/y] : E[\mathsf{succ}(c)/u]\end{array}}$$

The empty type

Formation rule.

$$\mathsf{0} : \mathsf{type}$$

No introduction rules.

Elimination rule.

$$\frac{c : \mathsf{0} \quad u : \mathsf{0} \vdash E : \mathsf{type}}{\mathsf{emptyrec}(c, (u)E) : E[c/u]}$$

No computation rules.

The unit type (I)

Formation rule.

$$1 : \mathsf{type}$$

Introduction rules.

$$* : 1$$

The unit type (II)

Elimination rules.

$$\frac{c : 1 \quad u : \mathsf{Bool} \vdash E : \mathsf{type} \quad d : E[*/u]}{\mathsf{unitrec}(c, (u)E, d) : E[c/u]}$$

Computation rule.

$$\frac{u : \mathsf{Bool} \vdash E : \mathsf{type} \quad d : E[*/u]}{\mathsf{unitrec}(*, (u)E, d) = d : E[*/u]}$$

The type of Boolean truth values (I)

Formation rule.

$$\mathsf{Bool} : \mathsf{type}$$

Introduction rules.

$$\mathsf{true} : \mathsf{Bool} \qquad \mathsf{false} : \mathsf{Bool}$$

The type of Boolean truth values (II)

Elimination rules.

$$\frac{c : \mathsf{Bool} \quad u : \mathsf{Bool} \vdash E : \mathsf{type} \quad d : E[\mathsf{true}/u] \quad e : E[\mathsf{false}/u]}{\mathsf{boolrec}(c, (u)E, d, e) : E[c/u]}$$

Notation. We will write $\mathsf{if}\ c\ \mathsf{then}\ d\ \mathsf{else}\ e$ for $\mathsf{boolrec}(c, (u)E, d, e)$.

Computation rules.

$$\frac{u : \mathsf{Bool} \vdash E : \mathsf{type} \quad d : E[\mathsf{true}/u] \quad e : E[\mathsf{false}/u]}{\mathsf{if}\ \mathsf{true}\ \mathsf{then}\ d\ \mathsf{else}\ e = d : E[\mathsf{true}/u]}$$

$$\frac{u : \mathsf{Bool} \vdash E : \mathsf{type} \quad d : E[\mathsf{true}/u] \quad e : E[\mathsf{false}/u]}{\mathsf{if}\ \mathsf{false}\ \mathsf{then}\ d\ \mathsf{else}\ e = e : E[\mathsf{false}/u]}$$

Sum types (I)

Formation rule.

$$\frac{A : \mathsf{type} \quad B : \mathsf{type}}{A + B : \mathsf{type}}$$

Introduction rules.

$$\frac{a : A}{\mathsf{inl}(A, B, a) : A + B} \qquad \frac{b : B}{\mathsf{inr}(A, B, b) : A + B}$$

Notation. We will write $\mathsf{inl}(a)$, $\mathsf{inr}(b)$ for $\mathsf{inl}(A, B, a)$ and $\mathsf{inr}(A, B, b)$.

Sum types (II)

Elimination rule.

$$\frac{c : A + B \quad u : A + B \vdash E : \mathsf{type} \quad x : A \vdash d : E[\mathsf{inl}(x)/u] \quad y : B \vdash e : E[\mathsf{inr}(y)u]}{\mathsf{case}(c, (u)E, (x)d, (y)e) : E[c/u]}$$

Computation rules.

$$\frac{a : A \quad u : A + B \vdash E : \mathsf{type} \quad x : A \vdash d : E[\mathsf{inl}(x)/u] \quad y : B \vdash e : E[\mathsf{inr}(y)u]}{\mathsf{case}(\mathsf{inl}(a), (u)E, (x)d, (y)e) = d[a/x] : E[\mathsf{inl}(a)/u]}$$

$$\frac{b : B \quad u : A + B \vdash E : \mathsf{type} \quad x : A \vdash d : E[\mathsf{inl}(x)/u] \quad y : B \vdash e : E[\mathsf{inr}(y)u]}{\mathsf{case}(\mathsf{inr}(b), (u)E, (x)d, (y)e) = e[b/y] : E[\mathsf{inr}(b)/u]}$$

List types (I)

Formation rule.

$$\frac{A : \mathsf{type}}{\mathsf{List}(A) : \mathsf{type}}$$

Introduction rules.

$$\frac{A : \mathsf{type}}{\mathsf{nil}(A) : \mathsf{List}(A)} \qquad \frac{a : A \qquad \ell : \mathsf{List}(A)}{\mathsf{cons}(A, a, \ell) : \mathsf{List}(A)}$$

Notation. We will write nil and $\mathsf{cons}(a, \ell)$ for $\mathsf{nil}(A)$ and $\mathsf{cons}(A, a, \ell)$.

List types (II)

Elimination rule.

$$\frac{\begin{array}{l}\ell : \mathsf{List}(A) \\ u : \mathsf{List}(A) \vdash E : \mathsf{type} \\ d : E[\mathsf{nil}/u] \\ x : A, y : \mathsf{List}(A), z : E[y/u] \vdash e : E[\mathsf{cons}(x,y)/u]\end{array}}{\mathsf{listrec}(\ell, (u)E, d, (x,y,z)e) : E[\ell/u]}$$

Computation rules.

$$\frac{\begin{array}{l}u : \mathsf{List}(A) \vdash E : \mathsf{type} \\ d : E[\mathsf{nil}/u] \\ x : \mathsf{List}(A), y : A, z : E[y/u] \vdash e : E[\mathsf{cons}(x,y)/u]\end{array}}{\mathsf{listrec}(\mathsf{nil}, (u)E, d, (x,y,z)e) = d : E[\mathsf{nil}/u]}$$

$$\frac{\begin{array}{l}\ell : \mathsf{List}(A) \\ a : A \\ u : \mathsf{List}(A) \vdash E : \mathsf{type} \\ d : E[\mathsf{nil}/u] \\ x : \mathsf{List}(A), y : A, z : E[y/u] \vdash e : E[\mathsf{cons}(x,y)/u]\end{array}}{\begin{array}{l}\mathsf{listrec}(\mathsf{cons}(a,\ell), (u)E, d, (x,y,z)e) = \\ \quad e[\ell/x, a/y, \mathsf{listrec}(\ell, (u)E, d, (x,y,z)e/u] : E[\mathsf{cons}(a,\ell)/u]\end{array}}$$

Identity types (I)

Formation rule.

$$\frac{A : \mathsf{type} \quad a : A \quad b : A}{\mathsf{Id}(A, a, b) : \mathsf{type}}$$

Introduction rule.

$$\frac{a : A}{\mathsf{refl}(A, a) : \mathsf{Id}(A, a, a)}$$

Notation. We will write $\mathsf{Id}_A(a, b)$ for $\mathsf{Id}(A, a, b)$, $\mathsf{refl}(a)$ for $\mathsf{refl}(A, a)$.

Identity types (II)

Elimination rule.

$$\frac{p : \mathsf{Id}_A(a,b) \quad x : A, y : A, u : \mathsf{Id}_A(x,y) \vdash E : \mathsf{type} \quad x : A \vdash d : E[x/y, \mathsf{refl}(x)/u]}{\mathsf{J}(a,b,p,(x,y,u)E,(x,y,u)d) : E[a/x, b/y, p/u]}$$

Computation rule.

$$\frac{a : A \quad x : A, y : A, u : \mathsf{Id}_A(x,y) \vdash E : \mathsf{type} \quad x : A \vdash d : E[x/y, \mathsf{refl}(x)/u]}{\mathsf{J}(a,a,\mathsf{refl}(a),(x,y,u)E,(x,y,u)d) = d[a/x] : E[a/y, \mathsf{refl}(a)/u]}$$

Dependent function types (I)

Formation rule.

$$\frac{x : A \vdash B : \mathsf{type}}{\Pi(x : A)B : \mathsf{type}}$$

Introduction rule.

$$\frac{x : A \vdash b : B}{\lambda(A, (x)B, (x)b) : \Pi(x : A)B}$$

Notation. We will write $\lambda x.b$ for $\lambda(A, (x)B, (x)b)$.

Special case. If $x \notin \mathrm{FV}(B)$, write $A \to B$ for $\Pi(x : A)B$.

Dependent function types (II)

Elimination rule.

$$\frac{f : \Pi(x : A)B \qquad a : A}{\mathsf{app}(f, a) : B[a/x]}$$

Computation rule.

$$\frac{x : A \vdash b : B \qquad a : A}{\mathsf{app}(\lambda x.b, a) = b[a/x] : B[a/x]}$$

Dependent pair types (I)

Formation rule.

$$\frac{x : A \vdash B : \mathsf{type}}{\Sigma(x : A)B : \mathsf{type}}$$

Introduction rule.

$$\frac{a : A \qquad b : B[a/x]}{\mathsf{pair}(A, (x)B, a, b) : \Sigma(x : A)B}$$

Notation. We will write $\mathsf{pair}(a, b)$ for $\mathsf{pair}(A, (x)B, a, b)$.

Special case. If $x \notin \mathrm{FV}(B)$, write $A \times B$ for $\Sigma(x : A)B$.

Dependent pair types (II)

Elimination rule.

$$\frac{c : (\Sigma x : A)B(x) \quad u : (\Sigma x : A)B(x) \vdash E : \mathsf{type} \quad x : A, y : B \vdash d : E[\mathsf{pair}(x, y)/u]}{\mathsf{split}(c, (u)E, (x, y)d) : E[c/u]}$$

Computation rule.

$$\frac{a : A \quad b : B[a/x] \quad u : (\Sigma x : A)B(x) \vdash E : \mathsf{type} \quad x : A, y : B \vdash d : E[\mathsf{pair}(x, y)/u]}{\mathsf{split}(\mathsf{pair}(a, b), (u)E, (x, y)d) = d[a/x, b/y] : E[\mathsf{pair}(a, b)/u]}$$

A type universe (I)

Formation rule.

$$\mathsf{U} : \mathsf{type}$$

Elimination rule.

$$\frac{A : \mathsf{U}}{\mathsf{T}(A) : \mathsf{type}}$$

A type universe (II)

Introduction and computation rules.

$$0^{\mathsf{U}} : \mathsf{U} \qquad \mathsf{T}(0^{\mathsf{U}}) = 0 : \mathsf{type}$$

$$1^{\mathsf{U}} : \mathsf{U} \qquad \mathsf{T}(1^{\mathsf{U}}) = 1 : \mathsf{type}$$

$$\mathsf{Bool}^{\mathsf{U}} : \mathsf{U} \qquad \mathsf{T}(\mathsf{Bool}^{\mathsf{U}}) = \mathsf{Bool} : \mathsf{type}$$

$$\mathsf{Nat}^{\mathsf{U}} : \mathsf{U} \qquad \mathsf{T}(\mathsf{Nat}^{\mathsf{U}}) = \mathsf{Nat} : \mathsf{type}$$

A type universe (III)

Introduction and computation rules.

$$\frac{A : \mathsf{U} \quad B : \mathsf{U}}{A +^{\mathsf{U}} B : \mathsf{U}} \qquad \frac{A : \mathsf{U} \quad B : \mathsf{U}}{\mathsf{T}(A +^{\mathsf{U}} B) = \mathsf{T}(A) + \mathsf{T}(B) : \mathsf{type}}$$

$$\frac{A : \mathsf{U}}{\mathsf{List}^{\mathsf{U}}(A) : \mathsf{U}} \qquad \frac{A : \mathsf{U}}{\mathsf{T}(\mathsf{List}^{\mathsf{U}}(A)) = \mathsf{List}(\mathsf{T}(A)) : \mathsf{type}}$$

A type universe (IV)

Introduction and computation rules.

$$\frac{A : \mathsf{U} \quad a : \mathsf{T}(A) \quad b : \mathsf{T}(A)}{\mathsf{Id}^{\mathsf{U}}(A, a, b) : \mathsf{U}} \qquad \frac{A : \mathsf{U} \quad a : \mathsf{T}(A) \quad b : \mathsf{T}(A)}{\mathsf{T}(\mathsf{Id}^{\mathsf{U}}(A, a, b)) = \mathsf{Id}(\mathsf{T}(A), a, b) : \mathsf{type}}$$

$$\frac{A : \mathsf{U} \qquad x : \mathsf{T}(A) \vdash B : \mathsf{U}}{\Pi^{\mathsf{U}}(A, (x)B) : \mathsf{U}} \qquad \frac{A : \mathsf{U} \qquad x : \mathsf{T}(A) \vdash B : \mathsf{U}}{\mathsf{T}(\Pi^{\mathsf{U}}(A, (x)B)) = \Pi(x : \mathsf{T}(A))\mathsf{T}(B) : \mathsf{type}}$$

$$\frac{A : \mathsf{U} \qquad x : \mathsf{T}(A) \vdash B : \mathsf{U}}{\Sigma^{\mathsf{U}}(A, (x)B) : \mathsf{U}} \qquad \frac{A : \mathsf{U} \qquad x : \mathsf{T}(A) \vdash B : \mathsf{U}}{\mathsf{T}(\Sigma^{\mathsf{U}}(A, (x)B)) = \Sigma(x : \mathsf{T}(A))\mathsf{T}(B) : \mathsf{type}}$$